motorway Documentation

Release 2.0.38

Plecto ApS

Dec 17, 2018

Contents

1 Pipeline **3**

2 Ramps **5**

3 Intersections **7**

4 Messages **9**

5 Contrib modules **11**
5.1 Amazon Kinesis 11
5.2 Amazon SQS 12

6 License **15**

Contents:

CHAPTER 1

Pipeline

class motorway.pipeline.**Pipeline**(*controller_bind_address='0.0.0.0:7007'*, *run_controller=True*, *run_webserver=True*, *run_connection_discovery=True*)

definition()

Extend this method in your *motorway.pipeline.Pipeline* subclass, e.g.:

```
class WordCountPipeline(Pipeline):
    def definition(self):
        self.add_ramp(WordRamp, 'sentence')
        self.add_intersection(SentenceSplitIntersection, 'sentence', 'word',␣
↪processes=2)
        self.add_intersection(WordCountIntersection, 'word', 'word_count',␣
↪grouper_cls=HashRingGrouper, processes=2)
        self.add_intersection(AggregateIntersection, 'word_count', grouper_
↪cls=HashRingGrouper, processes=1)
```

run()

Execute the entire pipeline in several sub processes.

CHAPTER 2

Ramps

class motorway.ramp.**Ramp**(*runs_on_controller=False*, *process_uuid=None*)

All messages must at some point start at a ramp, which ingests data into the pipeline from an external system or generates data it self (such as random words in the tutorial)

failed(*_id*)

Called when a message failed somewhere in the pipeline. The message might not be entirely finished processing at this point and this function might be called multiple times.

Parameters **_id** – The id of the message that failed

next()

This function is called continuously by the ramp.

> **Warning:** Do not block this for a long time or make a while True loop inside it. Betwen every *motorway.ramp.Ramp.next()* run, some essential operations are run, including receiving acks from the motorway.controller.Controller

Yield *motorway.messages.Message* instance

should_run()

Subclass to define rules whether this tap should run or not. Mainly used for ensuring a tap only runs once across the network

Returns bool

success(*_id*)

Called when a message was successfully ack'ed through the entire pipeline.

Parameters **_id** – The id of the message that was successful

CHAPTER 3

Intersections

class motorway.intersection.**Intersection**(*process_uuid=None*)

Intersections receive messages and generate either:

- A spin-off message

Spin-off messages will keep track of the state of the entire message tree and re-run it if failed. This means that if you want to re-run the message all the way from the ramp in case of an error, you should make a spin-off message.

Message.new(message, {

{ 'word': 'hello', 'count': 1

}, grouping_value='hello'

})

- A brand new message

The message will be created with the intersection as producer. The intersection will not receive feedback if it is successful or not and hence will not be re-tried in the case of an error.

Message(uuid.uuid4()

process(*message*)

This function is called continuously by the intersection.

Yield *motorway.messages.Message* instance

Parameters **message** – *motorway.messages.Message* instance or `list()` if using *motorway.decorators.batch_process()*

receive_messages(*context=None*, *output_stream=None*, *grouper_cls=None*)

Continously read and process using _process function

motorway.decorators.**batch_process**(*wait=5*, *limit=100*)

CHAPTER 4

Messages

class motorway.messages.**Message**(*ramp_unique_id*, *content=None*, *ack_value=None*, *controller_queue=None*, *grouping_value=None*, *error_message=None*, *process_name=None*, *producer_uuid=None*, *destination_endpoint=None*, *destination_uuid=None*)

Parameters

- **ramp_unique_id** – the unique message ID delivered back upon completion to the ramp
- **content** – any json serializable content
- **grouping_value** – String that can be used for routing messages consistently to the same receiver

Returns

ack(*time_consumed=None*)
Send a message to the controller that this message was properly processed

fail(*error_message=''*, *capture_exception=True*)
Send a message to the controller that this message failed to process

classmethod from_message(*message*, *controller_queue*, *process_name=None*)

Parameters

- **message** – Message dict (converted from JSON)
- **controller_queue** –
- **process_name** – UUID of the process processing this message (as string)

Returns

classmethod new(*message*, *content*, *grouping_value=None*, *error_message=None*)

Creates a new message, based on an existing message. This has the consequence that it will be tracked together and the tap will not be notified until every message in the chain is properly ack'ed.

Parameters

- **message** – Message instance, as received by the intersection
- **content** – Any value that can be serialized into json
- **grouping_value** – String that can be used for routing messages consistently to the same receiver

send_control_message(*controller_queue*, *time_consumed=None*, *process_name=None*, *destination_endpoint=None*, *destination_uuid=None*, *sender=None*)

Control messages are notifications that a new message have been created, so the controller can keep track of this particular message and let the ramp know once the entire tree of messages has been completed.

This is called implicitly on yield Message(_id, 'message')

Parameters **process_name** – UUID of the process processing this message (as string)

CHAPTER 5

Contrib modules

These are add-ons which is shipped with motorway, but not a part of the "core"

5.1 Amazon Kinesis

The Kinesis ramp is by far the most advanced available. It actually mimicks the behavior of the Amazon Kinesis Client Library but doesn't depend on Java like KCL.

The interface is very simple, just subclass *motorway.contrib.amazon_kinesis.ramps.KinesisRamp* and add the attribute "stream_name" according to the name you used for the stream in AWS.

Similarly, there is an intersection which allows you to "dump" content into a Kinesis stream. It works the exact same way.

class motorway.contrib.amazon_kinesis.ramps.**KinesisRamp**(*shard_threads_enabled=True*, ***kwargs*)

can_claim_shard(*shard_id*)

Determine whether or not a given shard can be claimed because of

1. It's currently not being processed by another process
2. It's unevenly balanced between the consuming nodes/workers

Parameters **shard_id** –

Returns bool

claim_shard(*shard_id*)

Atomically update the shard in DynamoDB

Parameters **shard_id** –

Returns bool

next()
This function is called continuously by the ramp.

> **Warning:** Do not block this for a long time or make a while True loop inside it. Betwen every *motorway.ramp.Ramp.next()* run, some essential operations are run, including receiving acks from the `motorway.controller.Controller`

Yield *motorway.messages.Message* instance

process_shard(*shard_id*)
Every shard (at startup) has an active thread that runs this function to either consume or wait to be ready to consume data from a shard

Parameters **shard_id** –

Returns

success(*_id*)
Called when a message was successfully ack'ed through the entire pipeline.

Parameters **_id** – The id of the message that was successful

class motorway.contrib.amazon_kinesis.intersections.**KinesisInsertIntersection**(***kwargs*)

process(*messages*)
wait 1 second and get up to 500 items Each PutRecords request can support up to 500 records. Each record in the request can be as large as 1 MB, up to a limit of 5 MB for the entire request, including partition keys. Each shard can support writes up to 1,000 records per second, up to a maximum data write total of 1 MB per second. This means we can run 2 intersections (2 x 500 records) submitting to the same shard before hitting the write limit (1000 records/sec) If we hit the write limit we wait 2 seconds and try to send the records that failed again, rinse and repeat If any other error than ProvisionedThroughputExceededException or InternalFailure is returned in the response we log it using loglevel error and dump the message for replayability instead of raising an exception that would drop the whole batch. So if you are going to use this intersection in production be sure to monitor and handle the messages with log level error! :param messages: :return:

5.2 Amazon SQS

class motorway.contrib.amazon_sqs.ramps.**SQSRamp**(**args*, ***kwargs*)

next()
This function is called continuously by the ramp.

> **Warning:** Do not block this for a long time or make a while True loop inside it. Betwen every *motorway.ramp.Ramp.next()* run, some essential operations are run, including receiving acks from the `motorway.controller.Controller`

Yield *motorway.messages.Message* instance

success(*_id*)
Called when a message was successfully ack'ed through the entire pipeline.

Parameters **_id** – The id of the message that was successful

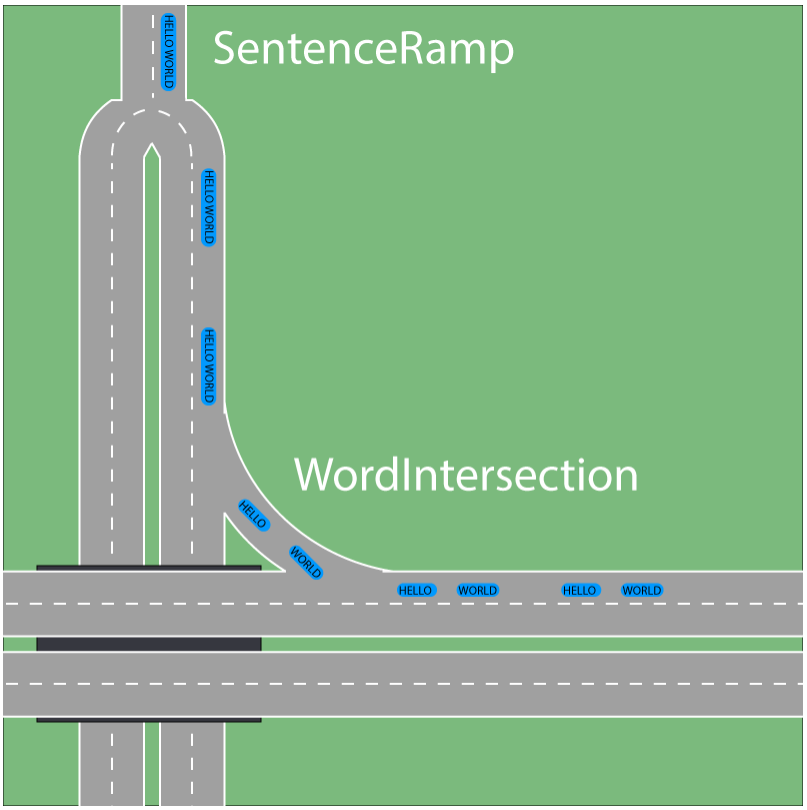
class motorway.contrib.amazon_sqs.intersections.**SQSInsertIntersection**(***kwargs*)

process(*message*)

This function is called continuously by the intersection.

Yield *motorway.messages.Message* instance

Parameters **message** – *motorway.messages.Message* instance or `list()` if using *motorway.decorators.batch_process()*



CHAPTER 6

License

Index

A

ack() (motorway.messages.Message method), 9

B

batch_process() (in module motorway.decorators), 7

C

can_claim_shard() (motorway.contrib.amazon_kinesis.ramps.KinesisRamp method), 11
claim_shard() (motorway.contrib.amazon_kinesis.ramps.KinesisRamp method), 11

D

definition() (motorway.pipeline.Pipeline method), 3

F

fail() (motorway.messages.Message method), 9
failed() (motorway.ramp.Ramp method), 5
from_message() (motorway.messages.Message class method), 9

I

Intersection (class in motorway.intersection), 7

K

KinesisInsertIntersection (class in motorway.contrib.amazon_kinesis.intersections), 12
KinesisRamp (class in motorway.contrib.amazon_kinesis.ramps), 11

M

Message (class in motorway.messages), 9

N

new() (motorway.messages.Message class method), 9
next() (motorway.contrib.amazon_kinesis.ramps.KinesisRamp method), 11
next() (motorway.contrib.amazon_sqs.ramps.SQSRamp method), 12
next() (motorway.ramp.Ramp method), 5

P

Pipeline (class in motorway.pipeline), 3
process() (motorway.contrib.amazon_kinesis.intersections.KinesisInsertInte method), 12
process() (motorway.contrib.amazon_sqs.intersections.SQSInsertIntersectio method), 13
process() (motorway.intersection.Intersection method), 7
process_shard() (motorway.contrib.amazon_kinesis.ramps.KinesisRamp method), 12

R

Ramp (class in motorway.ramp), 5
receive_messages() (motorway.intersection.Intersection method), 7
run() (motorway.pipeline.Pipeline method), 3

S

send_control_message() (motorway.messages.Message method), 10
should_run() (motorway.ramp.Ramp method), 5
SQSInsertIntersection (class in motorway.contrib.amazon_sqs.intersections), 13
SQSRamp (class in motorway.contrib.amazon_sqs.ramps), 12
success() (motorway.contrib.amazon_kinesis.ramps.KinesisRamp method), 12
success() (motorway.contrib.amazon_sqs.ramps.SQSRamp method), 12
success() (motorway.ramp.Ramp method), 5